Implicit Reduced Involutive Forms and Their Application to Engineering Multibody Systems

Wenqin Zhou[1], David J. Jeffrey[1], Gregory J. Reid[1], Chad Schmitke[2], and John McPhee[2]

[1] Department of Applied Mathematics, The University of Western Ontario, London, Ontario, Canada N6A 5B7
[2] Systems Design Engineering, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Abstract. The RifSimp package in Maple transforms a set of differential equations to Reduced Involutive Form. This paper describes the application of RifSimp to challenging real-world problems found in engineering design and modelling. RifSimp was applied to sets of equations arising in the dynamical studies of multibody systems. The equations were generated by the Maple package Dynaflex, which takes as input a graph-like description of a multibody mechanical system and generates a set of differential equations with algebraic constraints. Application of the standard RifSimp procedure to such Differential Algebraic Equations can require large amounts of computer memory and time, and can fail to finish its computations on larger problems.

We discuss the origin of these difficulties and propose an Implicit Reduced Involutive Form to assist in alleviating such problems. This form is related to RifSimp form by the symbolic inversion of a matrix. For many applications such as numerically integrating the multibody dynamical equations, the extra cost of symbolically inverting the matrix to obtain explicit RifSimp form can be impractical while Implicit Reduced Involutive Form is sufficient.

An approach to alleviating expression swell involving a hybrid analytic polynomial computation is discussed. This can avoid the excessive expression swell due to the usual method of transforming the entire input analytic differential system to polynomial form, by only applying this method in intermediate computations when it is required.

1 Introduction

A principal goal of multibody dynamics is the automatic generation of the equations of motion for a complex mechanical system, given a description of the system as input [14]. After generation, the set of equations must be analyzed or solved. Commercial programs exist that can generate and integrate such systems of equations. Adams, Dads and Working Model are examples of such products, and they are in widespread use in the automotive, aerospace and robotics industries [15]. These programs have many strengths, in particular they can handle systems containing many bodies (up to 100), but they have drawbacks.

H. Li, P. J. Olver and G. Sommer (Eds.): IWMM 2004, LNCS 3519, pp. 31–43, 2005.

For multibody systems, the general form of the dynamic equations is

$$M(t,q,\dot{q})\ddot{q} + \Phi_q^T \lambda = F(t,q,\dot{q}) \ , \tag{1}$$

$$\Phi(t,q) = 0 \ . \tag{2}$$

Here, q is a vector of generalized co-ordinates, $M(t,q,\dot{q})$ is the mass matrix, Φ is a vector of the constraint equations and λ is a vector of Lagrange multipliers [17, 18]:

$$\Phi = \begin{bmatrix} \Phi^1 \\ \vdots \\ \Phi^\ell \end{bmatrix} \ , \qquad \Phi_q = \begin{bmatrix} \Phi^1_{q_1} & \cdots & \Phi^1_{q_n} \\ \vdots & \vdots & \vdots \\ \Phi^\ell_{q_1} & \cdots & \Phi^\ell_{q_n} \end{bmatrix} \ , \qquad \lambda = \begin{bmatrix} \lambda^1 \\ \vdots \\ \lambda^\ell \end{bmatrix} \ .$$

Because the programs are purely numerical, it is difficult to check or comprehend the basic equations they have generated, and it is difficult to obtain analytic insight into the equations' properties. Also, when used for simulation, the programs are inefficient because the equations are effectively re-assembled at each time step, and the numerical assembly may include many multiplications in which one of the terms is 0 or 1. As a consequence, these programs are not well suited to real-time simulations and virtual reality, and, because of their large size, they cannot be downloaded onto the microprocessors that are typically used in real-time controllers [15, 3].

In contrast to numerically based programs, packages such as DYNAFLEX use symbolic programming to generate the equations of motion in a completely analytical form [16]. This approach offers several advantages [11]. The structure of the equations is easily obtained and manipulated, allowing one to gain a physical insight into the system; the equations can be easily exchanged with other researchers or engineers, something crucial to communication between different design groups; and real-time simulations are facilitated.

However, symbolic packages also have drawbacks. The equations generated by DYNAFLEX are usually too complicated to be solved symbolically. Even numerical solution is often difficult, inefficient or even impossible, because the equations are Differential Algebraic Equations (DAE), which typically are of second order but of high differential index. Also, the number of bodies that these programs can handle is not as large as for the numerically based programs.

Therefore, it is natural to develop methods for symbolically pre-processing the output of programs such as DYNAFLEX so that the output has desirable features such as being in simplified canonical form and including all constraints. Since any consistent initial value must satisfy all constraints, the inclusion of all constraints is a necessary condition for stating an existence and uniqueness theorem for such systems. The statement of such a theorem is another desirable feature for the output of such methods. Such features enable the consistent initialization of numerical solution procedures and can facilitate the identification of analytical solutions. In this paper we discuss how the RIFSIMP package can be used for the symbolic simplification of ODE and PDE systems and return canonical differential forms. It has the following features [9, 21]:

- Computation with polynomial nonlinearities.
- Advanced case splitting capabilities for the discovery of particular solution branches with desired properties.
- A visualization tool for the examination of the binary tree that results from multiple cases.
- Automatic generation of an existence and uniqueness theorem for the system.
- Algorithms for working with formal power series solutions of the system.

Applying RifSimp to the equations output by Dynaflex has the benefit of symbolically and automatically generating all special cases, through the RifSimp case-split options [9, 21]. In a full case analysis, some cases can be very complicated while others can be simple enough to be solved analytically. The canonical form generated by RifSimp is of low (0 or 1) differential index [1, 10] which is suitable for the application of numerical solvers. An important option with RifSimp is the possibility of excluding special cases that are known to be not of interest. Thus if we know that a special case, say $m = 0$, is of no physical interest, then we can append the *inequation* $m \neq 0$ to the input system [9, 21].

Application of the RifSimp package to multibody systems revealed that it has difficulty handling large systems generated by Dynaflex. The symptoms are excessive time and memory requirements. It is a well-known effect in computer algebra that these are linked, in that a computation that overflows physical memory will cause the operating system to swap memory to disk. The swapping, however, essentially brings the system to a halt. There are a number of contributing factors to the growth in time and memory, as will be described below. Therefore, if one wants to handle industrial-scale problems, one must modify the RifSimp approach. The modification we introduce here is the possibility of relaxing the requirements on the canonical form.

We remark that these are common difficulties encountered during the application of computer algebra methods to obtain canonical forms (e.g. such as Gröbner Bases for systems of multi-variate polynomials). An underlying idea in this paper is that many application may not require the full potency of canonical simplification (canonicity can be very expensive); and it is important to explore weaker non-canonical forms when they may achieve the objective in the given application.

2 Two Examples of Mechanical Systems

Two examples will be used to illustrate the application of computer algebra methods to multi-body systems.

The three dimensional top example considered in the paper, is an example of a small open-loop system. Other examples of open loop systems include robot arms or similar devices with a free end and a fixed end. The slider-crank mechanism considered in the paper, is an example of a small closed-loop system. Another typical example of a closed loop is a piston turning a crank through connecting rods, so that there are constraints on both the crank and the pis-

ton. Simple textbook problems in dynamics use *ad hoc* choices of co-ordinate systems to produce simple systems of equations without constraints modelling the problems. But this method is usually not possible with complicated systems, which are automatically generated by packages such as DYNAFLEX and constraint equations can not be eliminated. In addition, the constraints introduce additional variables (essentially Lagrange multipliers) into the equations, representing the forces they exert.

2.1 Open Loop: Three-Dimensional Top

In this classic problem, the top is an axisymmetric body that can precess about a vertical (Z) axis, nutate about a rotated X axis, and spin about a body-fixed symmetry axis, see figure 1. The top is assumed to rotate without slipping on the ground; this is modelled by placing a spherical (ball-and-socket) joint at O. DYNAFLEX automatically generates co-ordinates using standard 3-1-3 Euler angles (ζ, η, ξ), which correspond to precession, nutation, and spin, respectively.

The top has a moment of inertia J about the symmetry axis, and A about an axis at O perpendicular to the symmetry axis. Two angles η (the angle of the axis of symmetry to the z axis), and ζ specify the orientation of the axis of symmetry of the top, while ξ specifies how a point on the top is moving relative to its axis of symmetry. There is a coordinate singularity when η is equal to 0 or π, which RIFSIMP will automatically detect as part of its case analysis. Coordinate singularities are ubiquitous in automatically generated mechanical systems, and their automatic detection is an important problem.

The dynamic equations (1,2) generated by DYNAFLEX are, after changing from DYNAFLEX-generated symbols to more conventional ones, as follows. For details, we refer to the DYNAFLEX Users Guide [16].

$$M = \begin{bmatrix} (A\sin^2\eta + J\cos^2\eta) & 0 & J\cos\eta \\ 0 & A & 0 \\ J\cos\eta & 0 & J \end{bmatrix}, \quad q = \begin{bmatrix} \zeta \\ \eta \\ \xi \end{bmatrix} \tag{3}$$

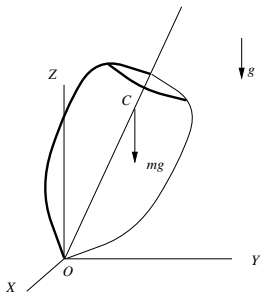

Fig. 1. The three-dimensional top. The centre of mass is at C and OC= l. Gravity acts in the $-Z$ direction

and

$$F = \begin{bmatrix} \sin(\eta)\left[2(J-A)\cos(\eta)\dot{\zeta} + J\dot{\xi}\right]\dot{\eta} \\ \sin(\eta)\left[(A-J)\cos(\eta)\dot{\zeta}^2 - J\dot{\xi}\dot{\zeta} + mgl\right] \\ J\sin(\eta)\dot{\eta}\,\dot{\zeta} \end{bmatrix} \tag{4}$$

The fact that the top is an open-loop system is reflected in the fact that DYNAFLEX has generated 3 equations for 3 unknowns.

2.2 Two-Dimensional Slider Crank

The slider crank is a simple example of a closed-loop system with $q^T = (\theta_1, \theta_2)^T$ where $\theta_1 = \theta_1(t)$ and $\theta_2 = \theta_2(t)$ are the angles shown in figure 2. The system is given by (1)–(2) where:

$$M = \begin{bmatrix} l_1^2(\frac{m_1}{4} + m_2 + m_3) + J_1 & -l_1 l_2 \cos(\theta_1 + \theta_2)(\frac{m_2}{2} + m_3) \\ -l_1 l_2 \cos(\theta_1 + \theta_2)(\frac{m_2}{2} + m_3) & l_2^2(\frac{m_2}{4} + m_3) + J_2 \end{bmatrix} \tag{5}$$

$$F = \begin{bmatrix} -l_1 g(\frac{m_1}{2} + m_2 + m_3)\cos\theta_1 - l_1 l_2 \dot{\theta_2}^2 \sin(\theta_1 + \theta_2)(\frac{m_2}{2} + m_3) \\ l_2 g(\frac{m_2}{2} + m_3)\cos\theta_2 - l_1 l_2 \dot{\theta_1}^2 \sin(\theta_1 + \theta_2)(\frac{m_2}{2} + m_3) \end{bmatrix} \tag{6}$$

and there is a single constraint equation between the angles:

$$\Phi = l_1 \sin\theta_1 - l_2 \sin\theta_2 = 0 \tag{7}$$

Therefore, $\Phi_q^T \lambda$ in (1) is given by $\Phi_q^T \lambda = \begin{pmatrix} l_1 \cos\theta_1 \\ -l_2 \cos\theta_2 \end{pmatrix} \lambda$. Note that in this example, λ is a scalar representing the normal reaction force of the constraint on the slider. In other words, in addition to generating the constraint equation (7), DYNAFLEX automatically generated the constraint force $\lambda(t)$. For both of these examples, the challenge now is to analyze the equations with computer algebraic methods such as RIFSIMP.

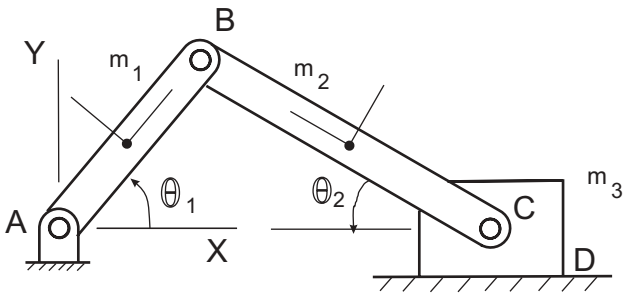

Fig. 2. The two-dimensional slider crank. The arm of length l_1 and mass m_1 rotates and causes the mass m_3 attached to the end of the arm of length l_2 to move left and right. Each arm has mass m_i and moment of inertia J_i, for $i = 1, 2$

3 Simplification Using RifSimp with Case Split

Given that symbolic algorithms such as Dynaflex exist for automatically producing the equations modelling multi-body systems, it is natural to exploit the further simplification and transformation of such systems using symbolic methods. In this section we discuss the simplification of such systems using the Maple algorithm `casesplit` which is part of RifSimp.

The theory underlying the Reduced Involutive Form given in [12] applies to systems of analytic nonlinear PDE in dependent variables $u_1, u_2, \ldots u_n$, which can be functions of several independent variables. While the general method applies to analytic systems, like most methods in computer algebra, the implemented algorithms apply to systems which are polynomial functions of their unknowns. This is to avoid well-known undecidability issues for classes of functions wider than polynomial or rational functions (e.g. there is no finite algorithm that can decide whether an analytic function is zero or non-zero at a point).

For the present application the only independent variable is time. In general the systems produced by Dynaflex are polynomial functions of sines and cosines of the angles $\theta_1(t), \theta_2(t), ..., \theta_n(t)$ between different components (arms and rotors etc).

One common method to convert such systems to rational form is the transformation

$$\cos\theta_j = x_j(t), \;\; \sin\theta_j = y_j(t), \;\; x_j^2 + y_j^2 = 1 \tag{8}$$

and another is the well-known Weierstrass transformation [5]:

$$\cos\theta_j = (1 - u_j(t)^2)/(1 + u_j(t)^2), \;\; \sin\theta_j = 2u_j(t)/(1 + u_j(t)^2) \tag{9}$$

where $whereu_j = tan(theta/2)$. If one solves for u_j, then the usual problems regarding choice of appropriate branch for the inverse arise. The transformation (9) has the advantage that the number of variables remains the same, and no new constraints are introduced. The transformation (8) has the disadvantage that additional constraints are introduced.

We will later discuss an alternative hybrid analytic-polynomial strategy. In that approach, the equations are manipulated in their original analytic form and conversions to polynomials by transformations such as those above are only used at intermediate computations and only for parts of the system which require the full algorithmic power of rational polynomial algebra. After resolving an analytic obstacle in this manner the inverse transformation yields the analytic form and the computation continues until the next algorithmic analytic obstacle is encountered.

RifSimp takes as its input a system of differential equations and a ranking of dependent variables and derivatives. Using its default ranking, RifSimp orders the dependent variables lexicographically and the derivatives primarily by total derivative order [9, 21]. For example for systems of ODE this default ranking is:

$$u_1 \prec u_2 \prec ... \prec u_n \prec u_1' \prec u_2' \prec ... \prec u_n' \prec u_1'' \prec ... \tag{10}$$

Each equation is then classified as being either leading linear or nonlinear, meaning linear or nonlinear in its highest derivative with respect to the ranking $\prec$.

RifSimp solves the leading linear equations for their highest derivatives until it can no longer find any such equations.

While solving explicitly for the highest derivatives, RifSimp splits cases based on the pivots (the coefficients of the leading derivatives) with which it divides. This yields a binary tree of cases.

Each leading-nonlinear equation (a so-called constraint) is differentiated and then reduced with respect to the current set of leading linear equations and then with respect to the leading nonlinear equations. A nonzero result means that this equation is a new constraint which should be appended to the system [9, 21].

For the current application if the solutions are 1 dimensional curves then each case output by the RifSimp algorithm has form:

$$v = f(t, w) \ , \tag{11}$$

$$g(t, w) = 0 \ , \tag{12}$$

$$h(t, w) \neq 0 \ . \tag{13}$$

Here v is the list of (highest-order) derivatives; w is a list of all derivatives, including dependent variables, lower in the ranking than v; g is a list of constraint equations; h is a list of inequations. From this form, it is possible to prove an existence and uniqueness theorem.

In particular, in our application, where there is a single independent variable t, the initial condition is $w(t^0) = w^0$ where the initial condition must satisfy the constraint $g(t^0, w^0) = 0$ and any inequations $h(t^0, w^0) \neq 0$ and this leads to a local analytic solution to the original system with this initial condition. Then the Existence and Uniqueness Theorem [12] states that there exists a local analytic solution satisfying the above initial conditions and inequations.

Application to Spinning Top

In order to apply RifSimp to equations (1,2) with M and F given by (3) – (4), we first convert the trigonometric functions to polynomials using the Weierstrass transformation, which is $\cos\eta = (1-u(t)^2)/(1+u(t)^2)$, $\sin\eta = 2u(t)/(1+u(t)^2)$. This yields a rational polynomial differential system. The resulting case tree produced by `casesplit` is surprisingly large, containing 24 cases, see figure 3.

It is important to understand the reasons for the many cases discovered by RifSimp, because for more complicated systems, the case analysis can become overwhelming. We first note that rigid-body mechanics is inherently complicated, and flexible body mechanics more so. The motion is mostly rotational, meaning that linear and angular momentum must be considered (introducing mass and moment-of-inertia parameters), and that the equations contain trigonometric functions. Beyond this, however, we note that one of the advantages of symbolic analysis for the engineer is the use of *symbolic* parameters for the masses, moments of inertia, lengths, etc. This is useful for their design studies, but it is well known in symbolic computing that the introduction of large numbers of symbols causes expression swell, slowdowns in the computation, and the occurrence of many special cases. Finally, for RifSimp, there is the problem that the program

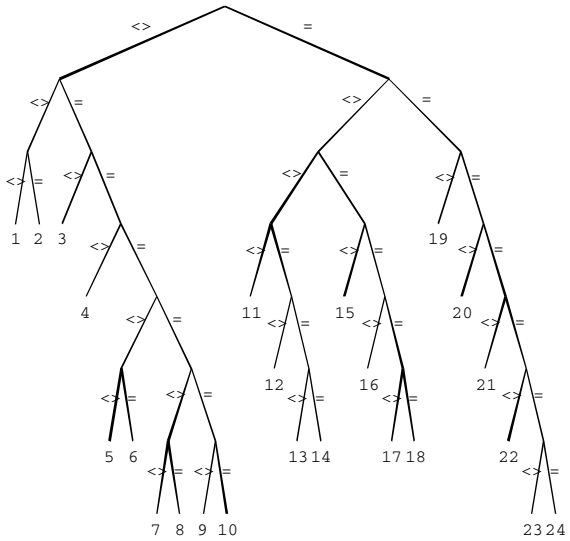

Fig. 3. Complete Case Tree for the 3D Top

assumes computation in the complex domain, whereas the engineering application only requires real variables. Thus the special cases identified in the complex plane are not relevant.

RifSimp has an input option that allows inequations to be appended to the system. These allow us to record physical facts such as $m \neq 0$, $g \neq 0$, etc. By recording as much information as possible in the list of inequations, the case tree can be significantly reduced. We find, for example, that the 24 cases in figure 3 can be reduced to 9. Amongst these special cases, RifSimp can identify dynamically degenerate cases. Examples are the cases of the top being oriented exactly vertically or exactly horizontally. In each case, one part of the precessional motion is not present. However, this strategy only delays the arrival of overwhelming expression swell, and we have therefore looked for an alternative to the standard RifSimp process.

4 Implicit Reduced Involutive Form Method

Two origins of expression swell for RifSimp are the following. If there are many equations containing many symbols, then inverting the matrix M to obtain explicit expressions for the $\ddot{q}$ results in division by many pivots that might be zero. After this, the reduction of equations modulo the existing equations is also difficult, because of the size of the equation set and the number of unknowns. From these observations we are led to seek a form weaker than a canonical solved form which is computationally feasible, while retaining as much of the power of RifSimp as possible. To this end, we introduce an *implicit* reduced involutive form.

Definition 4.1 [Implicit Reduced Involutive Form]: *Let $\prec$ be a ranking. A system $L = 0, N = 0$ is said to be in implicit reduced involutive form if there exist derivatives $r_1, ..., r_k$ such that L is leading linear in $r_1, ..., r_k$ with respect to $\prec$ (i.e. $L = A[r_1, \cdots, r_k]^T - b = 0$) and*

$$[r_1, \cdots, r_k]^T = A^{-1}b\ ,\ \ N = 0\ ,\ \ \det(A) \neq 0 \tag{14}$$

is in reduced involutive form.

This form is of interest, since computing A^{-1} on examples can be very expensive. Sometimes implicit rif-form can be obtained very cheaply, just by appropriate differentiation of the constraints.

Example 4.1 [Spinning Top]: In this case the system has form $M\ddot{q} = F$ and this is in implicit reduced involutive form provided $\det(M) \neq 0$. In general implicit reduced involutive form can be regarded as a cheap way of obtaining some but not all of the cases resulting from a system (e.g. the cases in this example with $\det(M) = 0$ are not covered).

Example 4.2 [General Multi-Body Systems]: To convert a system of general form (1,2) with non-trivial constraints to implicit reduced involutive form one would have to at least differentiate the constraints twice. Carrying this out we obtain

$$M\ddot{q} + \Phi_q^T\lambda = F(t, q, \dot{q}) \tag{15}$$

$$D_t^2\Phi = \Phi_q\ddot{q} + H\dot{q} + 2\Phi_{tq}\dot{q} + \Phi_{tt} = 0 \tag{16}$$

$$D_t\Phi = \Phi_q\dot{q} + \Phi_t = 0 \tag{17}$$

$$\Phi(t, q) = 0 \tag{18}$$

where $\Phi_{tq} = \frac{\partial\Phi_q}{\partial t}$, $\Phi_{tt} = \frac{\partial^2\Phi}{\partial t^2}$ and

$$H = \begin{bmatrix} \sum_i \Phi^1_{q_1q_i}\dot{q}_i & \cdots & \sum_i \Phi^1_{q_nq_i}\dot{q}_i \\ \vdots & \vdots & \vdots \\ \sum_i \Phi^\ell_{q_1q_i}\dot{q}_i & \cdots & \sum_i \Phi^\ell_{q_nq_i}\dot{q}_i \end{bmatrix}$$

We now show:

Theorem 4.1 *Consider the ranking $\prec$ defined by $q \prec \dot{q} \prec \lambda \prec \ddot{q} \prec \dot{\lambda} \prec \dddot{q} \cdots$ where the dependent variables q, λ are ordered lexicographically $q_1 \prec q_2 \prec ...$ and $\lambda_1 \prec \lambda_2 \prec ....$ The systems (15, 16, 17, 18) are in implicit rif-form with A, b, $[r_1, ..., r_k]^T$ in Definition 4.1 given by:*

$$A = \begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix}\ ,\ \ b = \begin{bmatrix} F(t, q, \dot{q}) \\ -H\dot{q} - 2\Phi_{tq}\dot{q} - \Phi_{tt} \end{bmatrix}\ ,\ \ [r_1, ..., r_k]^T = \begin{pmatrix} \ddot{q} \\ \lambda \end{pmatrix} \tag{19}$$

and $N = \{\Phi = 0,\ \Phi_q\dot{q} + \Phi_t = 0\}$, $\det(A) \neq 0$.

Proof: Set $N = \{\Phi = 0, D_t\Phi\} = \{\Phi = 0, \Phi_q\dot{q} + \Phi_t = 0\}$ in Theorem 4.1. Differentiating again yields $D_t^2\Phi$ given by (16).

Rewriting the system (15,16) in matrix form yields A, b, $[r_1, ..., r_k]^T$ in (19).

It remains to verify that $D_t\psi$ when reduced first with respect to L and then with respect to N yields zero for each $\psi \in N$. First $D_t\Phi = \Phi_q\dot{q} + \Phi_t$ is unaltered by reduction with respect to L and then reduces to zero with respect to N (since it is already in N). Next $D_t(\Phi_q\dot{q} + \Phi_t)$ is given in (16) and reduces to zero on simplification with respect to L (since it is a member of L). Note that we are working with analytic systems. Here as described in Rust [12] reduction to zero means detection as member of the analytic ring of functions (with coefficients again analytic functions) generated by the members of N. In general this is not algorithmic, but for multi-body systems by using one of the transformations to polynomial form, it can be converted into a polynomial ideal membership question which can be answered algorithmically, then transformed back to the analytic form. Here however the detection is trivial and does not require such techniques.

Again, we can note that implicit reduced involutive form easily obtained non-degenerate cases corresponding to $\det(A) \neq 0$. To determine whether a case is empty or not requires the analysis of whether there are any common solutions satisfying $N = 0$ and $\det(A) \neq 0$. This is a purely algebraic problem, which can be resolved in the worse case by applying one of the transformations to rational polynomial form. In that form one of the standard methods of commutative algebra (e.g. triangular sets) can be applied. Alternatively one can use some of the techniques of the new area of numerical algebraic geometry such as the methods of Sommese, Verschelde and Wampler [19]. That method determines so-called *generic* points on components of the variety determined by $N = 0$. Substitution of these points into A and application of some technique from numerical linear algebra (e.g. the singular value decomposition) can determine if $\det(A) \neq 0$.

A full analysis would have to consider the more difficult cases with $\det(A) = 0$. Indeed higher index problems (index > 2) yield $\det(A) = 0$ and further differentiations of the constraints need to be carried out to obtain implicit reduced involutive form.

Example 4.3 [Slider-Crank]: The example of the two-dimensional slider crank described above exactly fits into the class being discussed. Notice that even this simple case generates 5 independent parameters: each arm has a mass and a moment of inertia and the slider has a mass. The computation of this example is much harder to achieve in reduced involutive form.

5 Conclusion and Future Work

The mechanical systems generated by programs such as Dynaflex mean that they are ideal for testing new algorithms for dealing with DAE. The underlying idea is that such directly physical systems should lead to insights and new techniques for such DAE.

The implicit forms obtained can be useful in the numerical solution of such systems. For example, the matrix A above yields a system of DAE which can be solved using an implicit numerical method (i.e. along a solution curve, the

constant matrix A evaluated at a certain time step is a constant matrix, which is inverted using stable numerical methods). Thus a very expensive symbolic (exact) inversion of a matrix has to be compared to the solution using LU decomposition at each step along the path. In many applications we stress that the repeated solution of these systems along the path, are *much cheaper than symbolically inverting the matrix once and then evaluating the solution along the path.* Finding a balance between paying the cost of symbolic simplification, on the one hand, and, on the other, finding ways of working with implicit representations is a subject of our ongoing research. This is important for example in being able to carry out real time simulations.

In some respects the method that we eventually are approaching is quite similar to that appearing in the literature (e.g. see Visconti [20]). The purpose of the article is to try to draw rigorous differential elimination approaches closer together with such methods. In addition we suggest the use of analytic systems to assist in efficiency (avoiding a full polynomialization of the problem, since this can increase the complexity of the problem). In our calculations full polynomialization led to many extra equations compared to the analytic approach. The total degree (which is a measure of the complexity of the system) was often dramatically increased by the transformations, and this was reflected in our experience with calculations.

Indeed it is quite surprising that some of the techniques in DAE have not produced analogous strategies in general differential elimination packages for ODE and PDE such as `diffalg` or the RifSimp package. Our article is an effort to try to bridge this gap. Indeed an interesting aspect of the article and the work was the interaction between the authors from mechanical engineering (McPhee and Schmitke) and those from computer algebra (Jeffrey, Reid and Zhou). It forced the computer algebraists to examine some of the underlying techniques and assumptions routinely made in computer algebra. For example the conversion of analytic systems to rational polynomial form, is almost automatic and unquestioned as desirable in computer algebra approaches. The restriction to polynomial or rational polynomial functions also arose historically in the largely algebraic earlier era of symbolic computation. But as indicated in this article such a conversion can lead to unnecessarily large expressions.

We briefly discuss and compare reduced involutive form [12, 21] with the coordinate independent involutive form of geometric PDE [8, 13]. (Geometric) involutive form, provided certain regularity conditions are satisfied, does not depend on the explicit form of the PDE, but instead on their locus in Jet Space. Reduced involutive form, although closely related to involutive form, is not always involutive but can simply be prolonged without eliminations to involutive form [7]. Implicit reduced involutive form is closer to involutive form than the coordinate dependent regular differential chains of differential algebra [6] and coordinate dependent reduced involutive form. Both these coordinate dependent forms depend on having systems triangularized or solved with respect to their leading derivatives in the given ranking. Roughly, the solved-form requirement is dropped in the introduction of implicit reduced involutive form.

Our planned work includes other strategies for controlling the generation of large expressions, since there will always be a desire on the part of design engineers to increase the number of bodies that can be modelled. One strategy for large expression management (LEM) has been described in [4]. The key idea is that large expressions are not arbitrary collections of terms, but contain a structure. An analogy can be drawn with the situation in the study of matrices arising in engineering applications: they almost always have a 'structure' to them. For example, they are banded, or otherwise sparse. By recognizing structure, we can solve larger problems. Returning to symbolic manipulation, we can note that simplification routines in computer algebra can cause a loss of structure, usually with the result that larger expressions are generated. A very simple example is the apparent simplification of $(1+x)^9-1$, where a computer system will expand the bracket in order to cancel the 1 from the expansion with the 1 outside the bracket. Using the tools developed in [4] and now incorporated into Maple, we can preserve the structure inherent in engineering equations, such as those described here.

References

1. U. Ascher and L. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM (1998).
2. B. Buchberger, G. E. Collins. *Computer Algebra Symbolic and Algebraic Computation*. Springer-Verlag (1983).
3. P. Rideau. *Computer Algegbra and Mechanics, The James Software*. Computer Algebra in Industry I. John Wiley (1993).
4. R. M. Corless, D. J. Jeffrey, M. B. Monagan, Pratibha. *Two Perturbation Calculations in Fluid Mechanics Using Large-Expression Management*, J. Symbolic Computation **11**, 1–17, (1996).
5. David A. Cox, John B. Little, Donal O'Shea. *Ideals, Varieties, And Algorithms*. Springer-Verlag (1997).
6. E. Hubert. *Factorization free decomposition algorithms in differential algebra*. J. Symbolic Computation 29: 641–662, (2000).
7. E. Mansfield. *A Simple Criterion for Involutivity*. Journal of the London Mathematical Society 54: 323–345, 1996.
8. J.F. Pommaret. *Systems of Partial Differential Equations and Lie Pseudogroups*. Gordon and Breach Science Publishers, Inc. (1978).
9. A.D. Wittkopf, G.J. Reid. *The Reduced Involutive Form Package*. Maple Software Package. First distributed as part of Maple 7. (2001).
10. G.J. Reid, P. Lin and A.D. Wittkopf. *Differential-Elimination Completion Algorithms for DAE and PDAE*. Studies in Applied Mathematics, **106**, 1–45, (2001).
11. Christian Rudolf. *Road Vehicle Modeling Using Symbolic Multibody System Dynamics*. Diploma Thesis, University of Waterloo in cooperation with University of Karlsruhe (2003).
12. C.J. Rust. *Rankings of Partial Derivatives for Elimination Algorithms and Formal Solvability of Analytic Partial Differential Equations*. Ph.D. Thesis, University of Chicago (1998).
13. W.M. Seiler. *Analysis and application of the formal theory of partial differential equations*. Ph.D. thesis, Lancaster University, (1994).

14. W. Schiehlen. *Multibody Systems Handbook*. Springer-Verlag (1990).
15. Pengfei Shi, John McPhee. *Symbolic Programming of a Graph-Theoretic Approach to Flexible Multibody Dynamics*; Mechanics of Structures and Machines, 30(1), 123-154 (2002).
16. Pengfei Shi, John McPhee. *DynaFlex User's Guide*, Systems Design Engineering, University of Waterloo (2002).
17. P. Shi, J. McPhee. *Dynamics of flexible multibody systems using virtual work and linear graph theory*. Multibody System Dynamics, 4(4), 355-381 (2000).
18. P. Shi, J. McPhee, G. Heppler. *A deformation field for Euler-Bernouli beams with application to flexible multibody dynamics*. Multibody System Dynamics, 4, 79-104 (2001).
19. A.J. Sommese, J. Verschelde, and C.W. Wampler. *Numerical decomposition of the solution sets of polynomial systems into irreducible components*. SIAM J. Numer. Anal. 38(6), 2022–2046 (2001).
20. J. Visconti. *Numerical Solution of Differential Algebraic Equations Global Error Estimation and Symbolic Index Reduction*. Ph.D. Thesis. Laboratoire de Modélisation et Calcul. Grenoble (1999).
21. A.D. Wittkopf. *Algorithms and Implementations for Differential Elimination*. Ph.D. Thesis. Simon Fraser University, Burnaby (2004).